

Communication et partage d'information

IUT Lyon 1

Yosra ZGUIRA

yosra.zguira@insa-lyon.fr

2016 - 2017

Partage d'information

- Les cookies
 - Espace de données disponible sur les navigateurs Web
- Les données de sessions / application
 - Espace de mémoire du Web Container
- Les JavaBeans
 - Outils de partage d'information Web
- Les bases de données
 - Requêtes sur base de données

Les cookies

Introduction (1/3)

- Les cookies sont des données envoyées par le serveur web au navigateur client.
- Les cookies représentent un moyen simple de stocker temporairement des informations chez un client, afin de les récupérer ultérieurement.
- Il s'agit de fichiers texte stockés sur le disque dur du client après réception d'une réponse HTTP contenant des champs appropriés dans l'en-tête.
- Les cookies font partie des spécifications du **protocole HTTP**.

Introduction (2/3)

- Le protocole HTTP permet d'échanger des messages entre le serveur et le client à l'aide de requêtes HTTP et de réponses HTTP.
- Les requêtes et les réponses HTTP contiennent des en-têtes permettant d'envoyer des informations particulières de façon bilatérale.
- Un de ces en-têtes est réservé à l'écriture de fichiers sur le disque: **les cookies**.

Introduction (3/3)

- L'en-tête HTTP réservé à l'utilisation des cookies s'appelle Set-Cookie, il s'agit d'une simple ligne de texte de la forme:

Set-Cookie : NOM=VALEUR ; domain=NOM_DE_DOMAINE ; expires=DATE

- Il s'agit d'une chaîne de caractères commençant par **Set-Cookie** : suivie par des paires clés-valeur sous la forme CLE=VALEUR et séparées par des virgules.
- L'API servlet de Java propose un objet permettant de gérer de façon quasi-transparente l'usage des cookies, il s'agit de l'objet *Cookie*.

L'objet Cookie (1/2)

- La classe *javax.servlet.http.Cookie* permet de créer un objet Cookie encapsulant toutes les opérations nécessaires à la manipulation des cookies.
- Ainsi, le constructeur de la classe Cookie crée un cookie avec un nom et une valeur initiaux passés en paramètre.
- Il est toutefois possible de modifier la valeur de ce cookie ultérieurement grâce à sa méthode *setValue()*.
- Conformément à la norme HTTP 1.1, le nom du cookie doit être une chaîne de caractères ne contenant aucun caractère spécial défini dans la RFC 2068 (Il vaut mieux donc utiliser des caractères alphanumériques uniquement).

L'objet Cookie (2/2)

- Les valeurs par contre peuvent inclure tous les caractères hormis les espaces ou chacun de ces caractères :

[] () = , " / ? : ;

Envoi du Cookie

- L'envoi du cookie vers le navigateur du client se fait grâce à la méthode **addCookie()** de l'objet *HttpServletResponse*.

```
void addCookie (Cookie cookie)
```

- Etant donnée que les cookies sont stockés dans les en-têtes HTTP, et que celles-ci doivent être les premières informations envoyées, la création du cookie doit se faire avant tout envoi de données au navigateur.
 - ➔ Le cookie doit être créé avant toute écriture sur le flot de sortie de la servlet.

```
Cookie MonCookie = new Cookie ("nom", "valeur");  
response.addCookie(MonCookie);
```

Récupération des Cookies du client

- Pour récupérer les cookies provenant de la requête du client, il suffit d'utiliser la méthode **getCookies()** de l'objet *HttpServletRequest*.

```
Cookie[ ] getCookies( )
```

- Cette méthode retourne un tableau contenant l'ensemble des cookies présents chez le client.
- Il est ainsi possible de parcourir le tableau afin de retrouver un cookie spécifique grâce à la méthode **getName()** de l'objet *Cookie()*.

Récupération de la valeur d'un Cookie (1/4)

- La récupération de la valeur d'un cookie se fait grâce à la méthode **getValue()** de l'objet *Cookie*.

```
String Valeur = Cookie.getValue( )
```

- **Exemple:** L'exemple suivant permet de récupérer la valeur d'un cookie spécifique (dont le nom est « Cookiecherche ») et d'afficher sa valeur sur le navigateur du client :

Récupération de la valeur d'un Cookie (2/4)

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import java.net.*;

public class AfficheMonCookie extends HttpServlet {

    public void doGet (HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {

        Cookie[] cookies = request.getCookies();
```

Récupération de la valeur d'un Cookie (3/4)

```
for(i=0; i < cookies.length; i++) {  
  
    Cookie MonCookie = cookies[i];  
  
    if (MonCookie.getName().equals("Cookiecherche")) {  
  
        String Valeur = cookies[i].getValue();  
  
    }  
}  
  
// écriture de la reponse  
response.setContentType("text/html");  
  
PrintWriter out = response.getWriter();
```

Récupération de la valeur d'un Cookie (4/4)

```
out.println("<html><head>");  
  
out.println("<title>Mon Cookie</title>");  
  
out.println("</head><body>");  
  
out.println("Voici la valeur de mon cookie : " + Valeur);  
  
out.println("</body></html>");  
  
}  
}
```

Ajouter des paramètres à un Cookie (1/3)

Méthode	Description
Cookie (String Name, String Value)	Constructeur créant un objet Cookie de nom <i>Name</i> et de valeur <i>Value</i>
String getDomain ()	Retourne le domaine pour lequel le cookie est valide
void setDomain (String Domain)	Définit le domaine pour lequel le cookie est valide
int getMaxAge ()	Retourne la durée de validité du cookie (en secondes)
void setMaxAge (int duree)	Définit la durée de validité du cookie (en secondes)
String getName ()	Retourne le nom du Cookie

Ajouter des paramètres à un Cookie (2/3)


Méthode	Description
String getPath()	Retourne le chemin pour lequel le cookie est valide
void setPath (String Chemin)	Définit le chemin pour lequel le cookie est valide
boolean getSecure()	Retourne <i>true</i> si le cookie doit être envoyé uniquement sur ligne sécurisée, sinon <i>false</i>
boolean setSecure (Boolean flag)	Définit si le cookie doit être envoyé sur une ligne sécurisée (SSL)
String getValue()	Retourne la valeur du cookie
void setValue (String Valeur)	Redéfinit la valeur du cookie

Ajouter des paramètres à un Cookie (3/3)

Méthode	Description
int getVersion ()	Retourne la version du cookie
void setVersion (int Version)	Définit la valeur du cookie

Sessions

Problématique (1/2)

- **Le protocole HTTP** est un protocole **non connecté** (*protocole sans états*, en anglais *stateless protocol*).
 - ✓ Chaque requête est traitée indépendamment des autres,
 - ✓ Aucun historique des différents requêtes n'est conservé,
 - ✓ Le serveur Web ne peut pas se souvenir de la requête précédente,
-  Ce qui est dommageable dans des utilisations telles que le e-commerce, pour lequel le serveur doit mémoriser les achats de l'utilisateur sur les différentes pages.

Problématique (2/2)

- Il s'agit donc de maintenir la cohésion entre l'utilisateur et la requête:
 - ✓ reconnaître les requêtes provenant du même utilisateur,
 - ✓ associer un profil à l'utilisateur,
 - ✓ connaître les paramètres de l'application (nombre de produits vendus, ...).



On appelle ce mécanisme de gestion des états le **suivi de session** (en anglais *session tracking*).

L'objet HttpSession

- API des servlets: Objet **HttpSession**
- **Principe:**
 - ✓ A chaque utilisateur est associé implicitement un objet utilisé par les servlets pour sauvegarder un ensemble d'objets (un panier par exemple).
 - ✓ Cet objet de type `HttpSession` permet de suivre l'activité d'un client sur [plusieurs pages](#).
 - ✓ Les requêtes provenant d'un même utilisateur sont associées à un même objet.
 - ✓ Il sert comme «conteneur» pour des informations persistantes.
 - ✓ Durée de vie limitée = celle de la session

Fonctionnement d'une session (1/2)

- A la première requête vers une application web :
 - ✓ un objet HttpSession est créé
 - ✓ ainsi qu'un identifiant unique pour cet objet
- L'identifiant est en général sauvegardé par un cookie appelé **JSESSIONID**
=> seul l'identifiant de session est envoyé au client.
- Grâce à cet identifiant, le serveur détermine l'objet session correspondant à la requête courante.
- A toute nouvelle requête mise par l'utilisateur, le cookie est transmis vers le serveur web par la méthode: **public String HttpSession.getId()**

Fonctionnement d'une session (2/2)

- La gestion des sessions se fait de la manière suivante :
 - ✓ Obtenir l'ID de session
 - Si **GET**: en regardant dans la requête
 - Si **POST**: en regardant dans les en-têtes HTTP
 - Sinon dans les cookies
 - ✓ Vérifier si une session est associé à l'ID
 - Si la session existe ➔ obtenir les informations

- Sinon

- Générer un *ID de Session*

- Si le navigateur du client accepte les cookies, ajouter un cookie contenant l'ID de session

```
//Création du cookie  
Cookie C = new Cookie("id","723884532");  
  
//Définition de la limite de validité du cookie  
C.setMaxAge(24*3600);  
  
//Envoi du cookie dans la réponse HTTP  
res.addCookie(C);
```

- Sinon ajouter l'ID de session dans l'URL

```
<a href="http://serveur/MyServlet/exemple?id=723884532"> Lien hypertexte</a>
```

- Enregistrer la session avec l'ID nouvellement créé

Obtenir une session (1/2)

- La méthode **getSession()** de l'objet *HttpServletRequest* permet de retourner la session relative à l'utilisateur.
- L'identification est faite de façon transparente par cookies ou réécriture d'URL:

```
HttpSession getSession (boolean create)
```

- Si create = true → créer une session relative à la requête.
- On peut tester s'il y a un objet session dans la requête:
 - Si la méthode **getSession(false)** retourne *null*, alors il n'y a pas de session et il faut créer une nouvelle.
- On peut toujours tester par la méthode **isNew()** si une session est nouvelle.

Obtenir une session (2/2)

- Etant donnée que les cookies sont stockés dans les en-têtes HTTP, et que celles-ci doivent être les premières informations envoyées, la méthode **getSession()** doit être appelée avant tout envoi de données au navigateur.
- La méthode doit être invoquée avant toute écriture sur le flot de sortie de la servlet.

```
public class Example extends HttpServlet {  
    public void doGet (HttpServletRequest request, HttpServletResponse response) throws ServletException,  
        IOException {  
        //Récupérer la session  
        HttpSession session = request.getSession(true);  
  
        .....  
        //Ecrire la réponse  
        out = response.getWriter();  
  
        .....  
    }  
}
```

Obtenir des informations d'une session

- L'information contenue dans un objet **HttpSession** est de la forme
(attribut: String, valeur: Object)(attribut,valeur)(attribut, valeur)...
- La méthode **getAttribute(String)**
 - ✓ permet d'extraire la valeur correspondante à un attribut donné dans l'objet session.
 - ✓ retourne un objet de type Object
 - ➔ il est nécessaire de faire le transtypage (*casting*) pour manipuler la valeur retournée.
- Exemple :

```
HttpSession session = request.getSession(true);  
//Récupérer l'age de l'utilisateur  
Age = (int) session.getAttribute("Age");
```
- Cette méthode retourne *null*, si l'attribut n'existe pas.

Stocker des informations dans une session

- La méthode **setAttribute(String, Object)**
 - ✓ permet d'insérer une paire (*attribut*, *valeur*) dans un objet de type **HttpSession**

```
Object setAttribute("cle", "valeur");
```

- ✓ Si l'attribut existe déjà dans l'objet session, sa valeur sera mise à jour.

```
Panier p = new Panier();  
session.setAttribute("shopping", p);
```

Supprimer des informations dans une session

- La méthode **removeAttribute(String)**
 - ✓ Permet de supprimer une paire (*attribut, valeur*) dans un objet de type HttpSession.
 - ✓ Exemple :

```
session.removeAttribute("Schopping");
```

Autres méthodes de HttpSession (1/2)

Méthode	Description
long getCreationTime()	Retourne l'heure de la création de la session
Object getAttribute (String Name)	Retourne l'objet stocké dans la session sous le nom <i>Name</i> , <i>null</i> s'il n'existe pas
String[] getValueNames()	Retourne un tableau contenant le nom de toutes les clés de la session
Object getValue (String Name)	Retourne l'objet stocké dans la session sous le nom <i>Name</i> , <i>null</i> s'il n'existe pas
void invalidate()	Supprime la session

Autres méthodes de HttpSession (2/2)

Méthode	Description
void putValue (String Name, Object Value)	Stocke l'objet <i>Value</i> dans la session sous le nom <i>Name</i>
void removeValue (String Name)	Supprime l'élément <i>Name</i> de la session
void setAttribute (String Name, Object Value)	Stocke l'objet <i>Value</i> dans la session sous le nom <i>Name</i>
int setMaxInactiveInterval (int interval)	Définit l'intervalle de temps maximum entre deux requête avant que la session n'expire
Enumeration getAttributeNames()	retourne les noms de tous les attributs dans un objet session
String getId()	retourne l'identificateur unique généré pour chaque session

JavaBean

Problématique (1/2)

- **Constats :**

- ✓ Ecrire un script **JSP** consiste essentiellement à :

- ✗ Placer des programmes constitués d'instructions **Java** à l'intérieur de balises **JSP**.

- Nécessité de connaître le langage **Java**

- ✓ La création d'un site web dynamique s'effectue en équipe:

- ✗ Le développeur **Java** n'est pas toujours celui qui conçoit la partie **HTML** du site.

- Eviter aux concepteurs **HTML** d'avoir à comprendre et à insérer de nombreuses lignes de code **Java** à l'intérieur des pages **HTML**.

Problématique (2/2)

- **Solution :**

- ✓ Séparer le **code Java** du code **HTML**.

- ✗ Placer le **code Java** à l'intérieur de composants ou modules indépendants ayant comme métier de fournir un traitement précis.

- d'où le nom de **composants métier** ou aussi **JavaBean**.

- ✗ Les concepteurs **HTML** intègrent ces composants métier à l'intérieur de leurs pages en utilisant une syntaxe bien précise.

- **Outils :**

- ✓ Le langage **JSP** facilite cette intégration en proposant aux concepteurs **HTML** de nouvelles **Balises** qui appellent ces composants métier.

La notion de composant métier (1/2)

- **Les composants métier sont fournis par les développeurs **Java** et ont pour objectif d'assurer une fonctionnalité particulière du site.**

✓ Exemple :

✗ Lors de la mise en place d'un site de commerce électronique, il est nécessaire de développer un module sécurisé dédié au paiement en ligne par carte bancaire.

➤ Ce module est un composant métier à part entière.

➤ Il est généralement développé par une société externe, spécialisée dans le développement de logiciels sécurisés.

➤ Il est livré, au final, avec des spécifications précisant les valeurs demandées en entrée et celles fournies en sortie du composant.

La notion de composant métier (2/2)

- **Les composants métier sont :**
 - ✓ réutilisables,
 - ✓ codés une seule fois,
 - ✓ mais utilisés un très grand nombre de fois.
- **Les applications reposant sur une « [architecture](#) » en composant:**
 - ✓ permettent d'identifier parfaitement le rôle de chacun :
 - ✗ constructeur [HTML](#),
 - ✗ ou développeur [Java](#).
 - ✓ sont très faciles à maintenir ou à déployer.

Qu'est ce qu'un JavaBean? (1/2)

- **Définition humoristique:**

- ✓ JavaBean = Java (un café)+Bean (un grain de café)

- ✗ Un ensemble de grains de café permet la fabrication de café.

- ✗ Chaque grain apporte sa contribution à l'arôme et à la saveur du café.

- **Définition informatique:**

- ✓ JavaBean = Une classe Java indépendante qui définit des propriétés (données) et des méthodes décrivant ses différents services.

- ✗ Un logiciel est conçu par assemblage des services offertes par les JavaBeans.

- ✗ Chaque JavaBean apporte sa contribution aux traitements du logiciel.

Qu'est ce qu'un JavaBean? (2/2)

- **JavaBeans vs Classe Java classique :**

- ✓ La grande différence entre un **JavaBean** et une classe **Java** réside dans la façon dont le **JavaBean** échange ses données avec un script **JSP**.
- ✓ Les propriétés ou les données d'un **JavaBean** sont des attributs nommées qui peuvent être lus ou changés depuis un script **JSP** grâce à des méthodes dites **d'accès**.
 - ✗ Méthode d'accès en écriture qui modifie le contenu d'une propriété,
 - ✗ Méthode d'accès en lecture qui ne font que consulter la valeur stockée à l'intérieur de la propriété.
- ✓ Ces méthodes d'accès seront appelées à l'aide de nouvelles **Balises JSP** utilisées par les concepteurs **HTML**.

Radiographie d'un JavaBean (1/9)

- Considérons un exemple de **JavaBean** appelé «**CoutDeRevient**» qui a pour métier de calculer le coût de la construction d'une voiture quel que soit le modèle.
- Pour construire ce composant métier, il faut :
 - ✓ définir tout d'abord les propriétés nécessaire à la réalisation de ce calcul,
 - ✗ Ce sont les données, les attributs, ou encore les membres de la classe.
 - ✓ puis définir les méthodes associées.
 - ✗ Définir toutes les opérations et traitements réalisables sur ces propriétés.
 - ✗ Ces opérations sont aussi appelées méthodes, services ou comportements.

Radiographie d'un JavaBean (2/9)

- **1- Définir les propriétés du JavaBean :**

- ✓ Prenons pour hypothèse qu'une voiture est composée de quatre roues, d'une carrosserie, d'un moteur ainsi que d'un volant.
- ✓ Pour calculer le coût de construction d'une voiture, nous devons connaître tout d'abord le prix de chaque élément d'une voiture et ensuite calculer le coût total de la voiture.
- ✓ Les propriétés du JavaBean **CoutDeRevient** sont alors :
 - ✗ `private float coutRoue;`
 - ✗ `private float coutCarrosserie;`
 - ✗ `private float coutMoteur;`
 - ✗ `private float coutVolant;`
 - ✗ `private float coutRevient;`

Toutes les propriétés sont déclarées « **private** » pour être sûre que leurs valeurs ne seront modifiées que par le composant métier lui-même.

Radiographie d'un JavaBean (3/9)

- **2- Définir les comportements du JavaBean :**

- ✓ Définir des méthodes d'accès en écriture pour initialiser ou modifier les valeurs des coûts de chaque élément de la voiture.

- ✗ `public void set CoutRoue(float valeur);`

- ✗ `public void set CoutCarrosserie(float valeur);`

- ✗ `public void set CoutMoteur(float valeur);`

- ✗ `public void set CoutVolant(float valeur);`

Radiographie d'un JavaBean (4/9)

- ✓ Définir des méthodes d'accès en lecture pour récupérer les valeurs des coûts de chaque élément de la voiture.

✗ `public float get CoutRoue();`

✗ `public float get CoutCarrosserie();`

✗ `public float get CoutMoteur();`

✗ `public float get CoutVolant();`

✗ `public float get CoutRevient();`

Radiographie d'un JavaBean (5/9)

✓ Définir des méthodes **invisibles** ou **métier** qui effectuent un travail interne.

✗ **private boolean** verifCout();

➤ Méthode pour vérifier la validité des données enregistrées dans les propriétés du JavaBean.

✗ **private float** calculCoutRevient();

➤ Méthode pour calculer le coût de revient total d'une voiture.



- Ces deux méthodes sont déclarés en mode «**private**» parce que elles réalisent un traitement internes au composant.

- Elles ne doivent jamais être appelées depuis l'extérieur.

Radiographie d'un JavaBean (6/9)

- **3 - suivre certaines règles d'écriture du JavaBean :**

- ✓ Un JavaBean est une classe Java qui doit implémenter l'interface `java.io.Serializable`.

- ✗ Les propriétés du JavaBean doivent pouvoir être sauvegardées pour être restituées ultérieurement.

- ➔ Le mécanisme utilisé est la **sérialisation**.

- ➔ Pour permettre d'utiliser ce mécanisme, le JavaBean doit implémenter l'interface `Serializable`.

Radiographie d'un JavaBean (7/9)

- ✓ Un **JavaBean** est un composant évolué de **SUN**, définie comme le standard des applications orientées objets codées en **Java**. Il a pour objectifs de **normaliser** la façon dont il est codé.
- ✗ Les méthodes d'accès en écriture doivent commencer par « **set** », suivi immédiatement du nom de la propriété concernée. La première lettre de la propriété devient une majuscule.
- ✗ C'est en nommant ainsi les méthodes d'accès que les propriétés seront accessibles depuis un programme **JSP**.

Radiographie d'un JavaBean (8/9)

- Exemple de code source complet du JavaBean **CoutDeRevient**:

CoutDeRevient.java	
<pre>import java.io.Serializable ; public class CoutDeRevient implements java.io.Serializable { //Définition des propriétés private float coutRoue; private float coutCarrosserie ; private float coutMoteur ; private float coutVolant ; private float coutRevient ; // Définition des méthodes d'accès en écriture //La roue public void set CoutRoue (float valeur) { coutRoue = valeur ; } // La carrosserie public void set CoutCarrosserie (float valeur) { coutCarrosserie = valeur; } // Le moteur public void set CoutMoteur (float valeur) { coutMoteur =valeur; } // Le volant public void set CoutVolant (float valeur) { coutVolant =valeur; }</pre>	<pre>// Définition des méthodes d'accès en lecture //La roue public float get CoutRoue () { Return coutRoue; } // La carrosserie public float get CoutCarrosserie () { Return coutCarrosserie; } // Le moteur public float get CoutMoteur () { Return coutMoteur; } // Le volant public float get CoutVolant () { Return coutVolant; } // Le coutRevient public float get CoutRevient () { Return coutrevient; }</pre>

Radiographie d'un JavaBean (9/9)

CoutDeRevient.java (suite)

```
// Définition des méthodes d'accès en lecture
// Le coût total de revient
public float getCoutRevient() {
    coutRevient= calculCoutRevient ();
    return coutRevient;
}

// Définition des méthodes "métier"
// Vérifier la validité des données
private boolean verifCout () {
    if (getCoutRoue()<0 || getCoutMoteur()<0 || getCoutCarrosserie() < 0 || getCoutVolant() < 0 ) { return
false ; }
    else {
        return true ;
    }
}

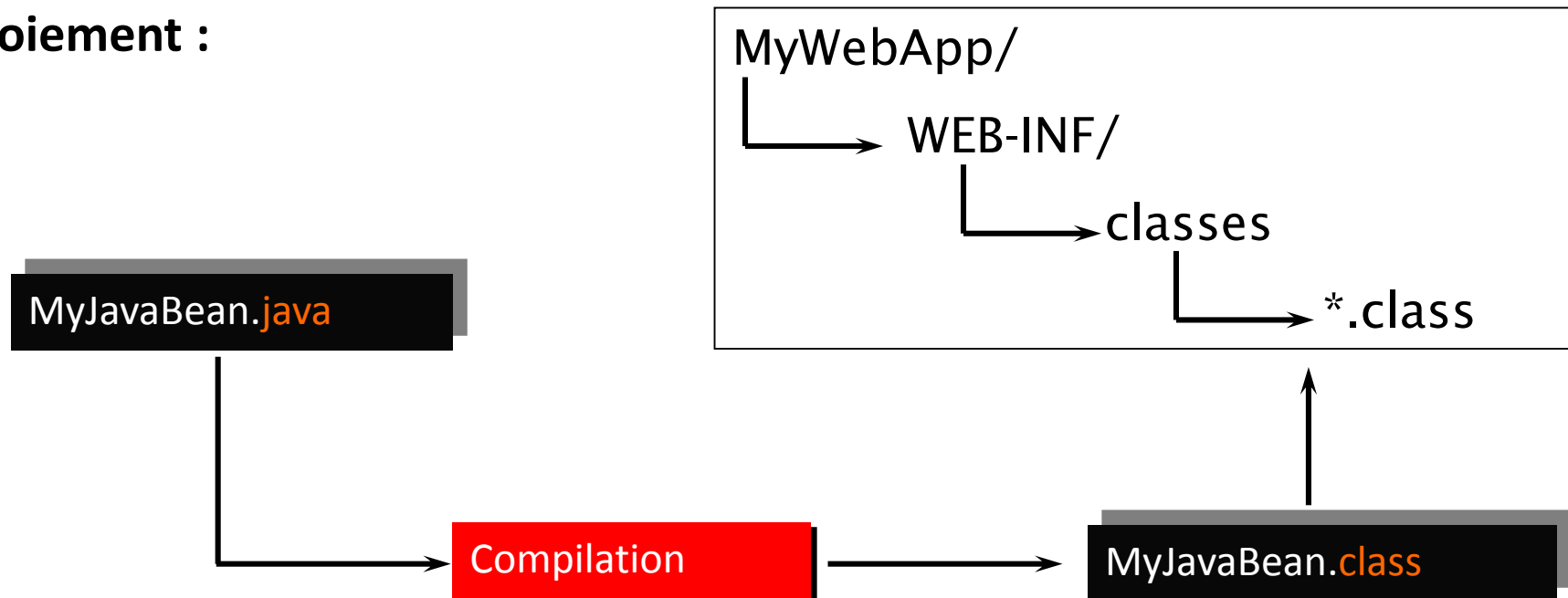
// Calculer le coût de revient
private float calculCoutRevient () {
    if (verifCout()) { return (getCoutRoue()*4+getCoutMoteur()+getCoutCarrosserie()+getCoutVolant()); }
    else {
        return -1f ;
    }
}
}
```

Comment rendre accessible un JavaBean?

- **Compilation :**

- ✓ Pour être utilisé, un **JavaBean** doit être compilé.

- **Le déploiement :**



Appeler un Bean depuis un JSP (1/5)

- L'étiquette **jsp:useBean**

✓ Syntaxe :

```
<jsp:useBean id="NomDeLobjet" class="NomDeLaClasse" />
```

- ✗ Cette instruction est l'équivalent à la déclaration d'un objet où l'on spécifie le type (nom de la classe) et la variable (nom de l'objet)
- ✗ Les attributs `class` et `id` permettent d'identifier le composant utilisé en indiquant le nom de la classe ainsi que son nom d'appel dans le JSP.

✓ Exemple :

```
<jsp:useBean id="myBean" class="CoutDeRevient" />
```

- ✗ Cette ligne est suffisante pour déclarer l'utilisation d'un `JavaBean` dans une JSP.

Appeler un Bean depuis un JSP (2/5)

- L'étiquette `jsp:setProperty`

✓ Syntaxe :

```
<jsp:setProperty name="NomDeLobjet" property="NomDePropriété" value="valeur" />
```

✗ Cette instruction permet de modifier la valeur d'une propriété d'un objet. Elle fait **automatiquement** appel à la méthode `SetSuiviDuNomDePropriété()` définie par la classe du **JavaBean**.

✓ Exemple :

```
<jsp:useBean id="myBean" class="CoutDeRevient" />
```

```
<jsp:setProperty name="myBean" property="coutRoue" value="100" />
```

✗ La propriété `coutRoue` de l'objet `myBean` vaut maintenant **100** grâce à la méthode `setCoutRoue(100)`.

Appeler un Bean depuis un JSP (3/5)

- L'étiquette `jsp:setProperty` (suite)

✓ Syntaxe :

```
<jsp:setProperty name="NomDeLobjet" property="NomDePropriété" />
```

✓ Exemple:

Formulaire

```
<input type="texte" size="10" name="coutRoue" />
```

```
<jsp:useBean id="myBean" class="CoutDeRevient" />
```

```
<jsp:setProperty name="myBean" property="coutRoue" />
```

- ✗ Lorsque l'utilisateur valide les valeurs saisies dans le formulaire (méthode **Post**) , la valeur `coutRoue` de ce formulaire est transmis **automatiquement** à la propriété `coutRoue` de l'objet `myBean`, grâce à la méthode **`setCoutRoue(coutRoue)`**.

Appeler un Bean depuis un JSP (4/5)

- L'étiquette `jsp:setProperty` (suite)

✓ Syntaxe :

```
<jsp:setProperty name="NomDeLobjet" property="NomDePropriété" param="valeur" />
```

✓ Exemple:

Formulaire

```
<input type =texte size=10 name =roue />
```

```
<jsp:useBean id ="myBean" class ="CoutDeRevient" />
```

```
<jsp:setProperty name ="myBean" property="coutRoue" param ="roue" />
```

- ✗ Lorsque l'utilisateur valide les valeurs saisies dans le formulaire(méthode **Post**) , la valeur **roue** de ce formulaire est transmis **automatiquement** à la propriété **coutRoue** de l'objet **myBean**, grâce à la méthode **setCoutRoue(roue)**.

Appeler un Bean depuis un JSP (5/5)

- L'étiquette **jsp:getProperty**

✓ Syntaxe:

```
<jsp:getProperty name="NomDeLobjet" property="NomDePropriété" />
```

✓ Exemple:

```
<jsp:useBean id="myBean" class="CoutDeRevient" />
```

```
<jsp:getProperty name="myBean" property="CoutRevient" />
```

✗ Cette instruction permet de déclencher **automatiquement** l'appel à la méthode d'accès en lecture **getCoutRevient()** de l'objet **myBean**.

Les bases de données

Rappel (1/4)

- **Création d'une base de données avec MySQL :**

- ✓ La création d'une BD passe par plusieurs étapes :

- ✗ Préparation de la base de données (BD)

- ✗ Description des tables avec des attributs et création de ces différentes tables

- Un nom de table s'écrit au pluriel, par exemple **Livres**, **Lecteurs**;

- Un nom commence toujours par une majuscule;

- Si le nom est composé, chaque mot qu'il contient commence par une majuscule;

- Un nom d'attribut commence par un préfixe correspondant aux trois premières lettres de la table en majuscule, suivi du caractère « _ », par exemple, pour la table **Lecteurs**, les noms des attributs sont **LEC_Nom**, **LEC_Prenom**, etc.

- ✗ Création des données effectives.

- ✗ Manipulation des données effectives

Rappel (2/4)

- **Exemple de préparation de la BD :**

- ✓ `USE mysql;`
- ✓ `DROP DATABASE IF EXISTS `nomdeBD`;`
- ✓ `CREATE DATABASE `nomdeBD`;`

- **Exemple de description et création des tables**

- ✓ `USE nomdeBD;`
- ✓ `DROP TABLE IF EXISTS `Lecteurs`;`
- ✓ `CREATE TABLE `Lecteurs` (
 LEC_NumLecteur char(16),
 LEC_NomLecteur char(16),
 LEC_PrenomLecteur char(16),
 LEC_Adresse char(80),
 LEC_Ville char(16),
 LEC_CodePostal char(10),
 PRIMARY KEY (`LEC_NumLecteur`)
);`

Rappel (3/4)

- **Exemple de Création des données effectives :**

- ✓ `INSERT INTO `lecteurs` VALUES ('216', 'Lamy', 'Eléna', '7 rue du Paradis', 'Paris', '75012');`

- ✓ `INSERT INTO `lecteurs` VALUES ('217', 'Théos', 'Pablo', '3 passage Secret', 'Paris', '75004');`

- **Exemple de Manipulation des données effectives**

- ✓ `SELECT * FROM Lecteurs;`

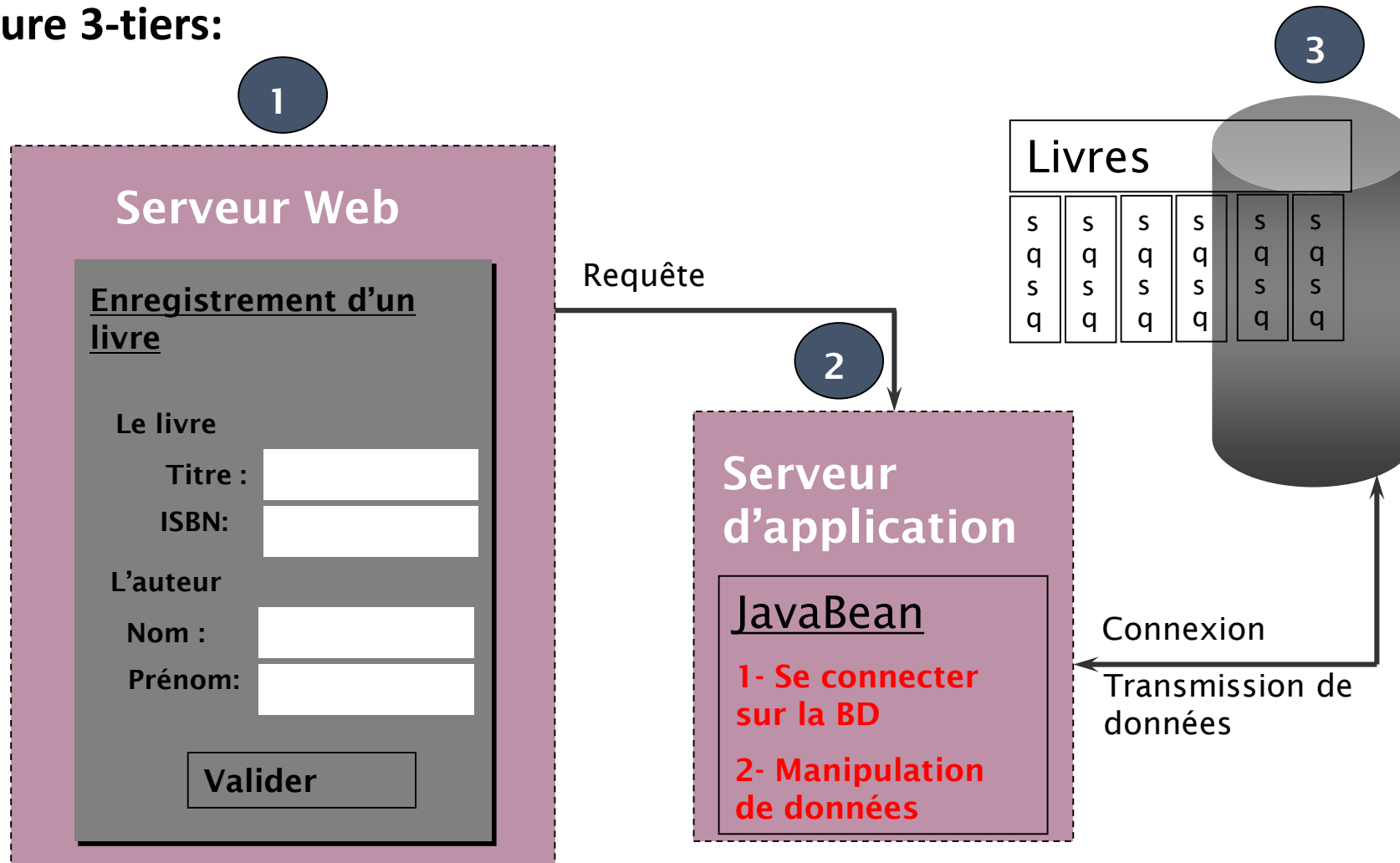
Rappel (4/4)

- **NB :**

- ✓ Dans le langage **MySQL**, une commande est traitée par le système à réception du caractère point-virgule « ; ».
- ✓ Toute commande doit se terminer donc par un point virgule.
- ✓ Le fait d'appuyer sur la touche Entrée en cours de saisie de la commande n'a aucune incidence sur le traitement de la commande.
- ✓ Toutes les étapes de création de la BD peuvent être regroupées dans un même fichier portant l'extension **.SQL**.

Communication JSP, JavaBean et BD MySQL (1/8)

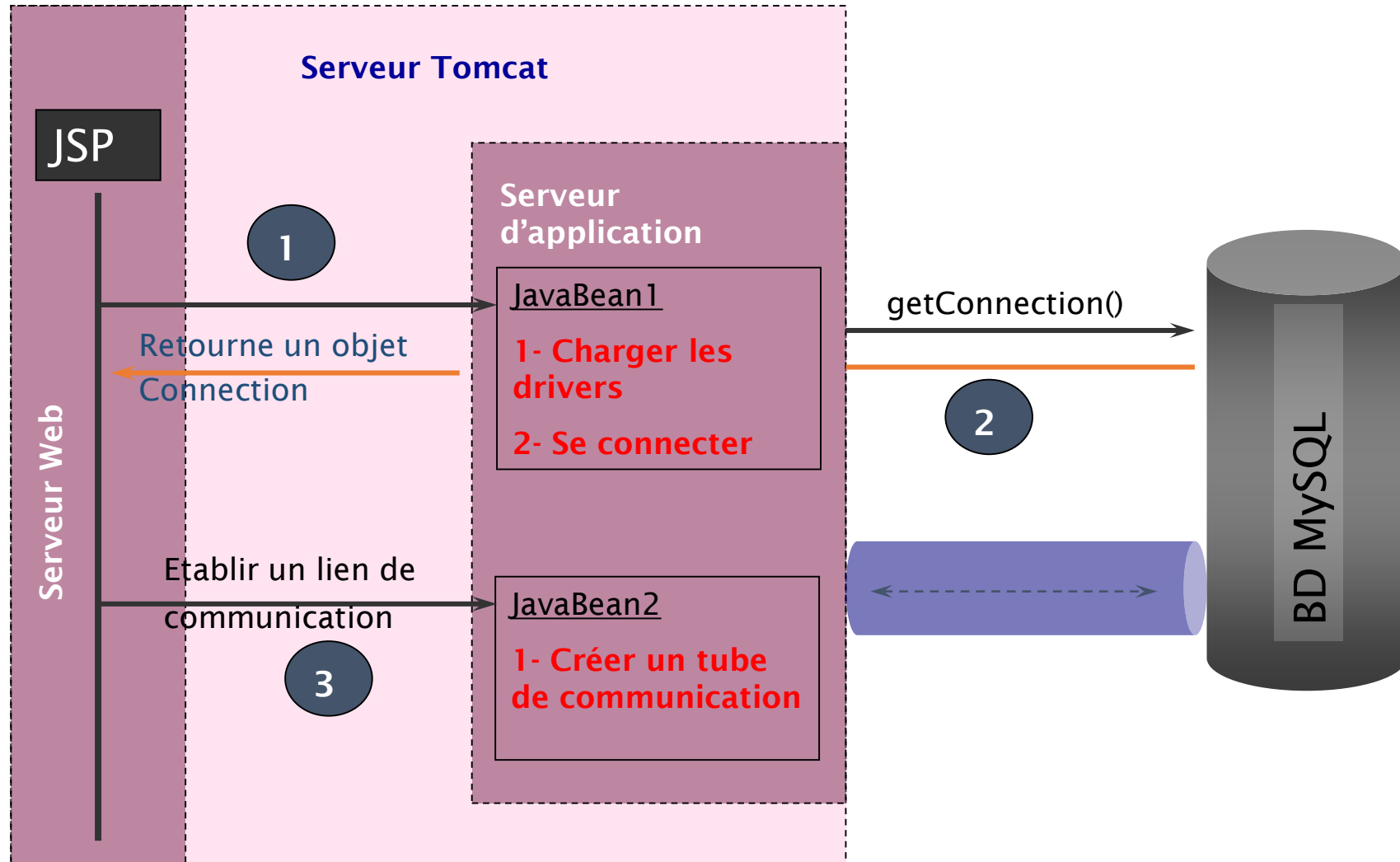
- Architecture 3-tiers:



Communication JSP, JavaBean et BD MySQL (2/8)

- La communication entre un **JSP**, un **JavaBean** et une **BD MySQL** nécessite plusieurs étapes :
 - 1- Reconnaissance de la base de données et création d'un objet **Connection**;
 - ✗ Le JavaBean doit charger en mémoire les drivers ou les pilotes qui lui permettent d'établir une connexion avec les BD.
 - 2- Mise en place d'un tube de communication à partir de l'objet **Connection**;
 - ✗ Une fois la connexion établie, le JavaBean doit créer un objet de liaison entre le **JSP** et la **BD MySQL**.
 - 3- Transmission de requêtes entre un JSP et une base de données via le tube.
 - ✗ Le JSP doit donc utiliser la connexion et le lien fournis par les deux composants JavaBean pour communiquer avec la BD.

Communication JSP, JavaBean et BD MySQL (3/8)



Communication JSP, JavaBean et BD MySQL (4/8)

- **Création d'un objet `Connection` :**

1- `Class.forName("com.mysql.jdbc.Driver");`

- ✗ Grâce à cette instruction, le pilote ou driver `JDBC` (Java DataBase Connector) relatif à la `BD MySQL` est chargé en mémoire.

2- `Connection cnx= DriverManager.getConnection(urlJdbc);`

- ✗ La méthode `getConnection` retourne une valeur de type `Connection` dans la variable `cnx`. Cet objet sert comme un lien de communication entre le `JSP` et la `BD MySQL`.
- ✗ La méthode `getConnection` est appelé via la classe `DriverManager` qui regroupe toutes les méthodes permettant la détection des pilotes et la création d'objet `Connection`.
- ✗ La méthode `getConnection` demande en paramètre où se trouve la `BD`, ainsi que le nom d'utilisateur et son mot de passe. Cette information est stockée sous la forme d'une chaîne de caractère nommée `urlJdbc`.
- ✗ `urlJdbc= "jdbc:mysql://" +hostname+ " : "+port+"/" +nomDeLaBase+ "?user= " +login+ "&password= " +password;`

Communication JSP, JavaBean et BD MySQL (5/8)

- Création d'un composant **JavaBean DBConnexion** :

```
DBConnexion.java

package Beans.BDConnection ;
import java.io.Serializable ;
import java.sql.*;

public class DBConnexion implements java.io.Serializable {

    //Définition des propriétés
    private String login;
    private String password ;
    private String hostname;
    private String port;
    private String nomDeLaBase;
    private Connection cnx;

    // Methodes d'accès en écriture
    // Enregistre le nom de login
    public void setLogin (String valeur) {
        login = valeur ;
    }

    // Enregistre le mot de passe
    public void setPassword (String valeur) {
        password = valeur ;
    }

    // Enregistre le nom du Host
    public void setHostname (String valeur) {
        hostname = valeur ;
    }

    // Enregistre le numéro de port
    public void setPort (String valeur) {
        port = valeur ;
    }

    // Enregistre le nom de la base de données
    public void setNomDeLaBase (String valeur) {
        nomDeLaBase = valeur ;
    }

    // Methodes d'accès en lecture
    public Connection getCnx() {
        if (etablirConnexion()) {
            return cnx;
        } else {
            return null;
        }
    }

    // Méthodes invisibles (métier)
    // Construit l'URL
    private String construireUrlJdbc() {
        String urlJdbc ;
        urlJdbc = "jdbc:mysql://" + hostname + ":" + port + "/" +
            nomDeLaBase ;
        urlJdbc = urlJdbc + "?user=" + login + "&password=" +
            password ;
        return urlJdbc ;
    }
}
```

Communication JSP, JavaBean et BD MySQL (6/8)

DBConnexion.java (suite)

```
// Etablit la connexion
private boolean etablierConnexion() {
    boolean statusConnexion = false;
    String urlJdbc ;
    try {
        Class.forName("com.mysql.jdbc.Driver");
        urlJdbc=construireUrlJdbc();
        cnx=DriverManager.getConnection(urlJdbc);
        statusConnexion = true;
    } catch ( Exception e ) {
        statusConnexion = false;
        System.out.println(e);
    }
    return statusConnexion;
}
```


Communication JSP, JavaBean et BD MySQL (7/8)

- **Création d'un tube de communication :**

- ✓ La mise en place de cette liaison s'effectue à travers un objet de type **Statement**.

- ✗ La classe **Statement** est une interface définie par **Java**.

- ✗ Un objet de type **Statement** est comprable à un tube bidirectionnel permettant de communiquer dans les deux sens **JSP-BD** et **BD-JSP**.

- ✗ Pour créer un objet **Statement**, il suffit de faire appel à la méthode **createStatement()** d'un objet de type **Connection**. Exemple :

- **Connection cnx** =

- **Statement lien** = **cnx.createStatement();**

Communication JSP, JavaBean et BD MySQL (8/8)

- Création d'un composant **JavaBean DBLien** :

DBLien.java

```
package Beans.BDConnection ;
import java.io.Serializable ;
import java.sql.*;

public class DBLien implements java.io.Serializable {
    //Définition des propriétés
    private Statement lien = null;

    // Methodes d'accès en lecture
    public Statement getLien(Connection cnx) {
        if (construireStatement(cnx)) {
            return lien;
        } else {
            return null;
        }
    }

    // Methodes invisibles
    private boolean construireStatement (Connection cnx) {
        boolean statusStatement = false;
        try {
            lien=cnx.createStatement();
            statusStatement = true ;
        } catch ( Exception e ) {
            statusStatement = false;
            System.out.println(e);
        }
        return statusStatement;
    }
}
```

Consulter ou modifier une BD MySQL (1/2)

- Modifier une **BD MySQL** a partir d'un **JSP**:

MyJSP.jsp

.....

.....

<%

//préparation de la requête

String commande = "INSERT INTO `Lecteurs` VALUES ('216', 'Lamy',
'Eléna', '7 rue du Paradis', 'Paris', '75012')";

lien.exeuteUpdate(commande);

%>

Consulter ou modifier une BD MySQL (2/2)

- Consulter une **BD MySQL** a partir d'un **JSP**:

MyJSP.jsp

```
.....  
<%  
//préparation de la requête  
String commande = "select * from Lecteurs";  
ResultSet rs;  
rs = lien.exeuteQuery(commande);  
If (rs!=null)  
{  
    while (rs.next()){  
        out.println("Nuémro Lecteur =" + rs.getString("LEC_NumLecteur"));  
        out.println("Nom Lecteur =" + rs.getString("LEC_NomLecteur"));  
        .....  
    }  
}%>
```

Exemple complet (1/3)

MyJSP.jsp

```
<!-- La page JSP manipule des objets de type Connection et Statement -->
<%@ page import="java.sql.*" %>

<!--// DBConnexion Bean-->
<jsp:useBean id="dbcnx" class="Beans.BDConnection.DBConnexion">

<!--// Code d'initialisation //-->
<jsp:setProperty name="dbcnx" property="login" value="admin"/>
<jsp:setProperty name="dbcnx" property="password" value=""/>
<jsp:setProperty name="dbcnx" property="hostname" value="localhost"/>
<jsp:setProperty name="dbcnx" property="port" value="3306"/>
<jsp:setProperty name="dbcnx" property="nomDeLaBase" value="test"/>
</jsp:useBean>

<!--// DBLien Bean -->
<jsp:useBean id="dblien" class="Beans.BDConnection.DBLien"/>

<html>
<body>
<table width=700><tr><td>
<h1><font face=arial>Test de Connexion à une Base de données </h1>
<h3><font face=arial>La connexion avec la base de données</h3>

<font face=arial size=2 color=black>1ère étape : Création d'une connexion vers la base de données :<br>
```

Exemple complet (2/3)

MyJSP.jsp

```
<%  
  // Récupération de la connexion  
  Connection cnx = dbcnx.getCnx();  
  if (cnx == null) {  
    out.println("<font color=red>Connexion impossible : " + cnx + "</font>");  
  } else {  
    out.println("<font color=green>Connexion établie : " + cnx + "</font>");  
  }  
%>  
  
<br><br>  
<font face=arial size=2 color=black>2ème étape : Création d'un tube de communication :<br>  
  
<%  
  // Création du tube de communication avec la base de données  
  Statement lien = dblien.getLien(cnx);  
  if (lien == null) {  
    out.println("<font color=red>Communication impossible : " + lien + "</font>");  
  } else {  
    out.println("<font color=green>Communication établie : " + lien + "</font>");  
  }  
%>  
</td></tr></table>  
</body>  
</html>
```

Exemple complet (3/3)

